CS1010 Programming Methodology

Week 4 and 5: C Basics and Functions

> People learn something every day, and a lot of times it's that what they learned the day before was wrong.
> ~*Bill Vaughan*

To students:

The success of discussion sessions hinges very much on (1) your **PREPARATION** beforehand, (2) your **ACTIVE PARTICIPATION** in class, and (3) after-class **REVISION**. Unless otherwise stated, you do not need to submit your work for grading.

Please cooperate with your DL to work towards a fruitful and enriching learning experience.

Due to time constraint, sometimes not all the questions in the Discussion Sheet will be discussed in class. Your DL has the discretion to choose the questions (or you may request your DL to discuss certain questions) or set his/her own questions for you. You may continue to discuss the questions on the IVLE forums after class.

Also, why limit yourself only to the exercises here? You may find exercises from other sources (books or the Internet) for your own practice.

I. C Basics and Exploration

Your DL may skip some questions in this section if you have explored them on your own and figured out the answers.

1. Spot the errors, bad programming practice and logic flaws in the following program. Compile the program and see if you understand the error messages.

```
#include <stdio.h>

int Main(void) {
float j, k, l;

scanf("%.2f", k);
k = l + 1.2;
printf("k = %.2f\n", k);
return 0;
}
```

2. The following programs work but are very badly written. Explain why and how would you improve them?

(a) Program to compute the volume of a cone.

```
#include <stdio.h>
int main(void) {
double a,s,d,f;
a=3.14159; scanf("%lf %lf",&s,&d);
f=1.0/3.0*a*s*s*d;
printf("%.2f\n",f);
return 0;}
```

(b)

```
#include <stdio.h>

int main(void) {
    // declare the int variables num1, num2 and num3
    float num1,           num2, num3;

// ask user to enter two values into num1 and num2
       printf("Enter two real numbers: ");
    scanf("%f %f", &num1, &num2);

num3 = 0.0; // initialise num3 to 0.0
// divide num1 by num2, then multiply the result by num2
// and then assign the result to num3
        num3 = (num1/num2) * num2;
     printf("num3 = %f\n", num3); // display value in num3

return 0; }
```

Run this program with your own input data. Do you always get the correct answer? See question 2.

Tip on vim: Do you know how to do global indentation of your C program in vim?
In command mode, type gg=G

3. Exploration.
   (a) Have you discovered that the program in Q2b does not always produce an accurate answer?

   Sometimes we want to examine the value of a variable for debugging purpose, or to check an intermediate result. What would you suggest for the program in Q2b?

   (b) Suppose you have added a **printf()** statement in the program in Q2b for the purpose of checking, test your program on these inputs:

   - `123.1 2.0`

   What do you observe? Do you know why?

4. Exploration: What are the outputs of this program?

```
printf("0.0/3.0 = %f\n", 0.0/3.0);
printf("3.0/0.0 = %f\n", 3.0/0.0);
printf("0.0/0.0 = %f\n", 0.0/0.0);
```

What if you try it on integers? You will get this error message if you try to perform 3/0:

```
Arithmetic Exception (core dumped)
```

This is a run-time error (an error that occurs during a run and not during compilation). What has happened is what we call a "core dump".

Tip: What can you do if you get a "core dump"?

Students often ask us, why does my program get a "core dump"? The answer is, there can be 101 reasons your program gets a "core dump". In the above example, it is due to a division-by-zero error. Hence, it is IMPOSSIBLE for us to tell you right away what happened, unless we take a look at your code.

A "core dump" happens when a program terminates abnormally (crashes). The system then dumps the content of the memory into a file called "core". This name comes from the days when magnetic core memory was used.

The file "core" is usually very huge, so it is advisable that you delete it (use UNIX command "rm core"). What follows, of course, is that you have to study your source code to find out what went wrong.

5. Exploration.

(a) Compile this program without and with the **–Wall** option. What do you observe? What is the output of the program?

```
#include <stdio.h>

int main(void) {
    int i;
    float f;
    double d;

    printf("i = %d; f = %f; d = %lf\n", i, f, d);
    return 0;
}
```

(b) What are the values assigned to the variables?

```
    int a, b, c;
    a = b = c = 12;
```

(c) What are the values assigned to the variables?

```
    int a, b, c;
    a = b + 2 = c + 1 = 5;
```

(d) What are the values assigned to the variables?

```
    float a;
    int b;
    double c;
    a = b = c = 12.98;
```

(e) What is the output of this program?

```
#include <stdio.h>

int main(void) {
    int a, b, c, d, e, f;

    a = 3;
    b = a++ + 10;
    printf("a = %d; b = %d\n", a, b);

    c = 3;
    d = ++c + 10;
    printf("c = %d; d = %d\n", c, d);

    e = f = 3;
    f *= 2 + e;
    printf("e = %d; f = %d\n", e, f);

    return 0;
}
```

(f) Spot the redundancy in this program.

```
    int x, y = 2;
    x = 0;
    x = y * y;
```

6. You learnt in **Unit 3 slide 22** that the algorithm to swap two integer variables `num1` and `num2` is as follows:

```
temp = num1;
num1 = num2;
num2 = temp;
```

Wilson came out with this alternative algorithm, and claimed that it is better because it does not require an additional variable `temp`:

```
num1 = num1 – num2;
num2 = num1 + num2;
num1 = num2 – num1;
```

Trace the algorithm on some test data. It appears that the algorithm is correct, but what is wrong with such an algorithm?

II. Functions

You have learnt functions that do not return any value (void functions) and functions that return one value (through the return statement).

7. (a) Why do we write modular programs? Compare the two programs on NRIC check code below. Why do you think the modular one is preferred?

```
// Non-modular
int main(void) {
   int number;
   char code;
   int dig1, dig2, dig3, dig4, dig5, …

   printf("Enter NRIC number: ");
   scanf("%d", &number);

   // determine check code
   . . .
   printf("Check code is %c\n", code);

   return 0;
}
```

```
// Modular
int main(void) {
   int number;
   char code;

   printf("Enter NRIC number: ");
   scanf("%d", &number);
   code = generateCode(number);
   printf("Check code is %c\n", code);

   return 0;
}

// This function …
char generateCode(int num) {
   char code;
   int dig1, dig2, dig3, dig4, dig5, …
   . . .
   return code;
}
```

(b) Why is the following function considered bad?

```
// This function ...
void generateCode(int num) {
   char code;
   int dig1, dig2, dig3, dig4, dig5, …
   . . .

   printf("Check code is %c\n", code);
}
```

8. Spot the errors in the program below and correct them. Keep the function **func()** below the **main()** function.

```
#include <stdio.h>

int main(void) {
    void func(5);
    void func(3-7);

    return 0;
}

void func(y) {
    if (y<0) {
        printf("Nothing\n");
    }
    else {
        printf("Something\n");
    }
}
```

9. Trace the program below manually. Determine the values of the variables `a`, `b`, and `c` in the **main()** function, and the parameters `a`, `b`, and `c` in the **confuse()** function at every step.

   What are the final values of `a`, `b`, and `c` in the **confuse()** function just before control returns to the **main()** function, and what are the final values of `a`, `b`, and `c` in the **main()** function?

```
#include <stdio.h>

int confuse(int, int, int);

int main(void) {
  int a = 6, b = 2, c = 5;
  a = confuse(c, b, a);

  return 0;
}

int confuse(int b, int a, int c) {
    a = b + c;
    c = a * b;

    return c - a + b;
}
```

10. Write a program **triangle.c** to request for data of 3 points on a 2-dimensional plane forming the vertices of a triangle, and compute the perimeter of the triangle. Each point is represented by two non-negative integers in the x-coordinate and y-coordinate.

    You may assume that the input data are always valid and they represent the vertices of a triangle.

    The perimeter is to be displayed in 2 decimal places.

    You program is to include a user-defined function called **distance()** that returns the distance (of **double** type) between two points. Your function may call some appropriate functions in math.h.

    A sample run of the program is given below. User's inputs are in bold.

```
Enter 1st point: 1 5
Enter 2nd point: 3 3
Enter 3rd point: 7 4
Perimeter of triangle = 13.03
```